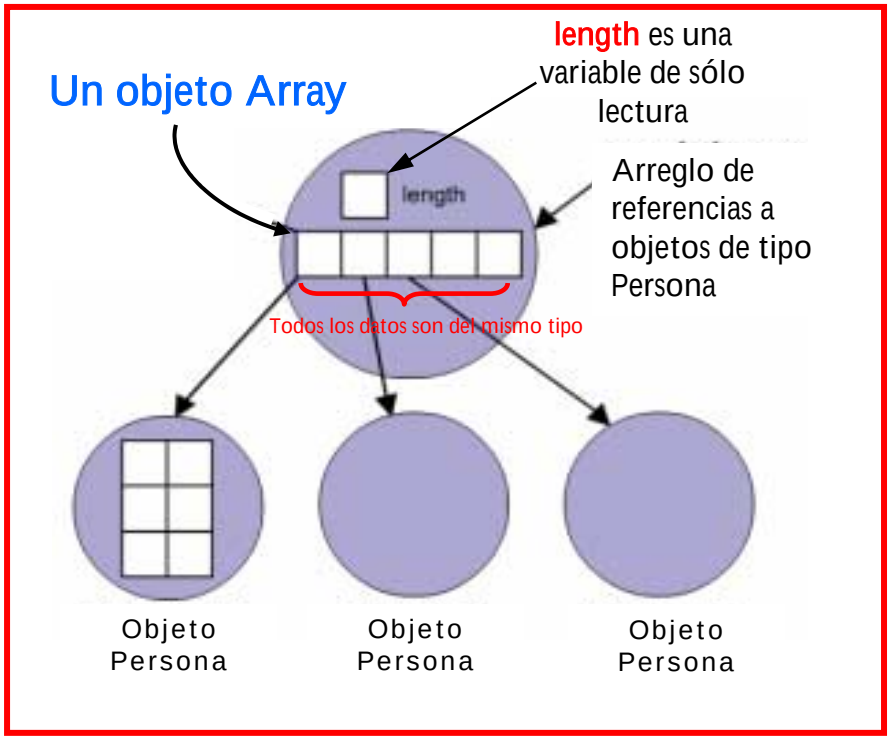
Arreglos y Colecciones

Lic. Claudia Queiruga (claudiaq@info.unlp.edu.ar) - Lic. Laura Fava (lfava@info.unlp.edu.ar)

Arreglos
Introducción

- Un objeto arreglo crea y guarda datos en un conjunto secuencial de ubicaciones de almacenamiento.
- Los arreglos son tipos de datos pre-construídos, que contienen una determinada cantidad (fija) de objetos del mismo tipo.
- Un arreglo es un objeto que hace referencia a muchos valores primitivos o a objetos, a través de una única variable.
- Los datos almacenados en un arreglo, se guardan en posiciones contiguas y son todos del mismo tipo.



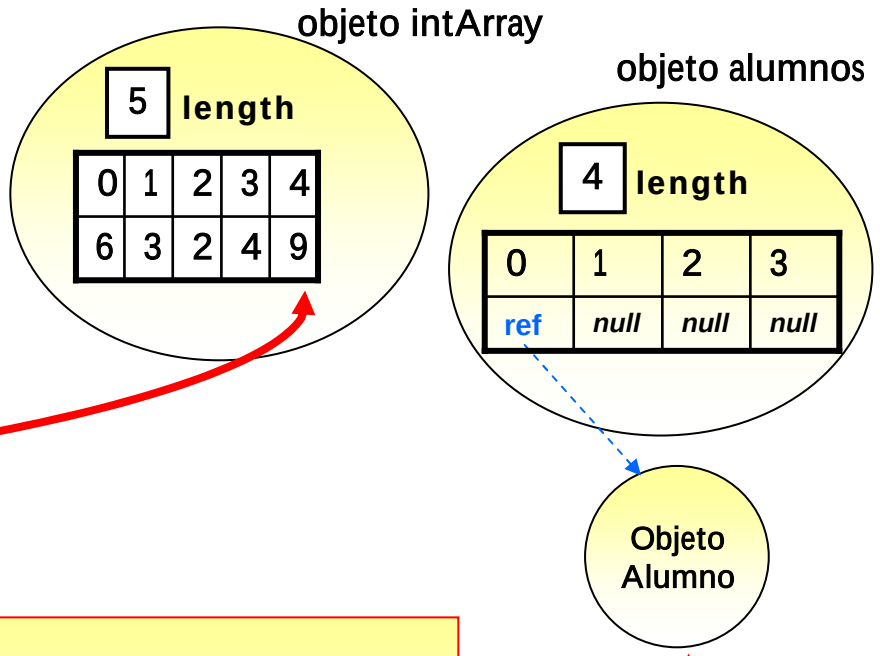
Arreglos

Arreglos de Primitivos y de Objetos

```
public class ArreglodePrimitivos {
 public static void main(String[] args){
   int[] intArray;
   intArray = new int[5];
   intArray[0] = 6;
   intArray[1] = 3;
   intArray[2] = 2;
   intArray[3] = 4;
   intArray[4] = 9;
 }}
```

La declaración crea la variable intArray, no el objeto arreglo.

El operador *new* crea el objeto arreglo, con la cantidad de elementos




```
public class ArreglodeObjetos {
 public static void main (String[] args){
  int numDeEstu;
  Alumno[] alumnos = new Alumno[4];
  for (int i = 0; i < alumnos.length;  i++) {
    String unNombre=Console.readLine("Ingrese el nombre:");
    String unCurso=Console.readLine("Ingrese el curso:");
    alumnos[i] = new Alumno(unNombre, unGrado);
  }
 }
}
```

Se declara e inicializa un objeto arreglo de Alumno. Cada elemento almacena *null*.

Se crea cada objeto Alumno y se almacena su referencia en la posición i-ésima del arreglo.

Si el objeto alumno, tuviera una variable de instancia nombre, se podría acceder: **`alumnos[o].nombre`**

Arreglos

Arreglos de Primitivos y de Objetos

- Los arreglos también se pueden inicializar en el momento de la declaración:

```
int[] cantDiasMes = {31,28,31,30,31,30,31,31,30,31,30,31};
String[] generos = {“acción”, “drama”, “Comedia”};
```

- También pueden definirse arreglos multidimensionales:

```
int [ ] [ ] scores = {{66, 78, 78, 89, 88, 90},
                      {76, 80, 80, 82, 90, 90},
                      {90, 92, 87, 83, 99, 94} };
```

```
int[][] scores= new int[3][6];
```

[] []	0	1	2	3	4	5
0	66	78	78	89	88	90
1	76	80	80	82	90	90
2	90	92	87	83	99	94

Primera dimensión hace referencia a la fila (alumno)

Segunda dimensión hace referencia a la columna (los exámenes)

Por ejemplo: scores[2,3] es 83, hace referencia al cuarto puntaje del tercer alumno

- Hay 2 excepciones comunes disparadas por mal manejo de arreglos:
  - **ArrayIndexOutOfBoundsException**: si se intenta acceder a una posición inválida del arreglo
  - **NullPointerException**: si se trata de acceder a un elemento del arreglo que no ha sido inicializado.

```
String[] palabras= new String[3];
String palabra = palabras[3];
```

```
String[] palabras;
palabras[0] = “Hola”;
```

Arreglos y Colecciones
Principales diferencias

- El JDK 1.2 introdujo un framework para colecciones de objetos, llamado Java Framework Collection. Este framework está dentro del paquete java.util

- Una colección es similar a un arreglo, en cuanto a que es un objeto simple, que representa a un conjunto de objetos, llamados elementos. Existen varias diferencias entre ellos:

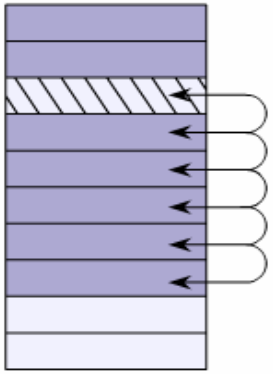
  - Las colecciones pueden manejar distintos tipos de objetos, los arreglos no.
  - Todas las colecciones almacenan objetos de tipo Object. Para recuperar un objetos desde una colección se requiere hacer un casting explícito del objeto a su tipo.
  - Los objetos de tipo Collection representan referencias a otros objetos. Los arreglos pueden contener datos primitivos y referencias a otros objetos, pero las colecciones solamente pueden contener referencias a objetos de tipo Object.
  - Los objetos Collection son más flexibles que arreglos: pueden crecer dinámicamente, pueden mantener a los elementos ordenados, pueden agregar/eliminar elementos en forma más eficiente.

Colecciones
Tecnologías de almacenamiento

Existen cuatro tecnologías de almacenamiento básicas disponibles para almacenar objetos: arreglo, lista enganchada, árbol y tabla de *hash*.

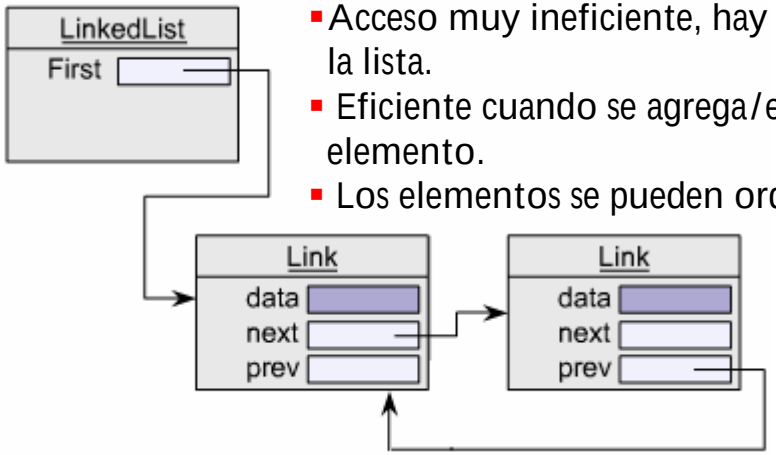
Arreglo

- Almacenamiento de valores de un único tipo.
- El acceso es muy eficiente.
- Es ineficiente cuando se agrega/elimina un elemento.
- Los elementos se pueden ordenar



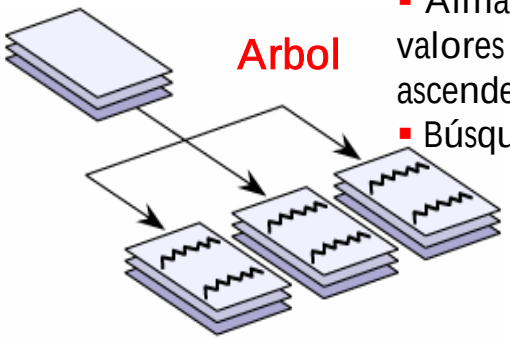
Lista Enganchada

- Acceso muy ineficiente, hay que recorrer la lista.
- Eficiente cuando se agrega/elimina un elemento.
- Los elementos se pueden ordenar.



Arbol

- Almacenamiento de valores en órden ascendente.
- Búsqueda eficiente.



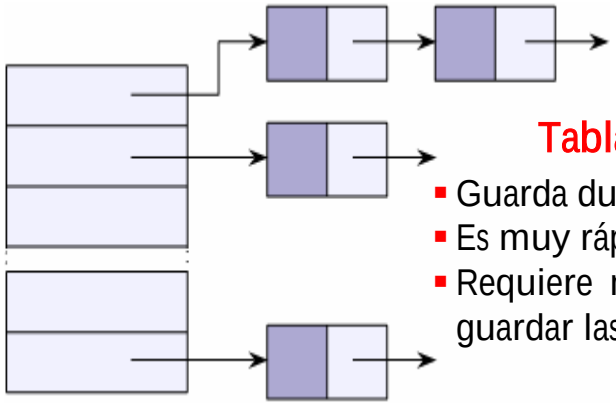
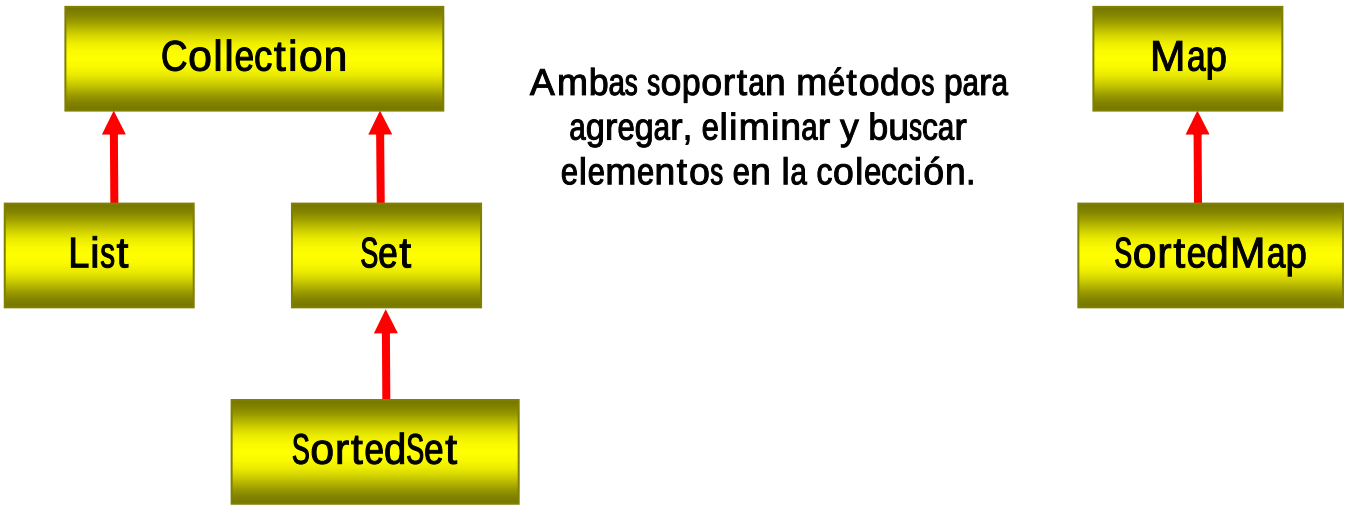


Tabla de *hash*

- Guarda duplas (clave, valor).
- Es muy rápido, accede por calve
- Requiere memeoria adicional para guardar las claves (tabla).

Colecciones
Tipos de Colecciones

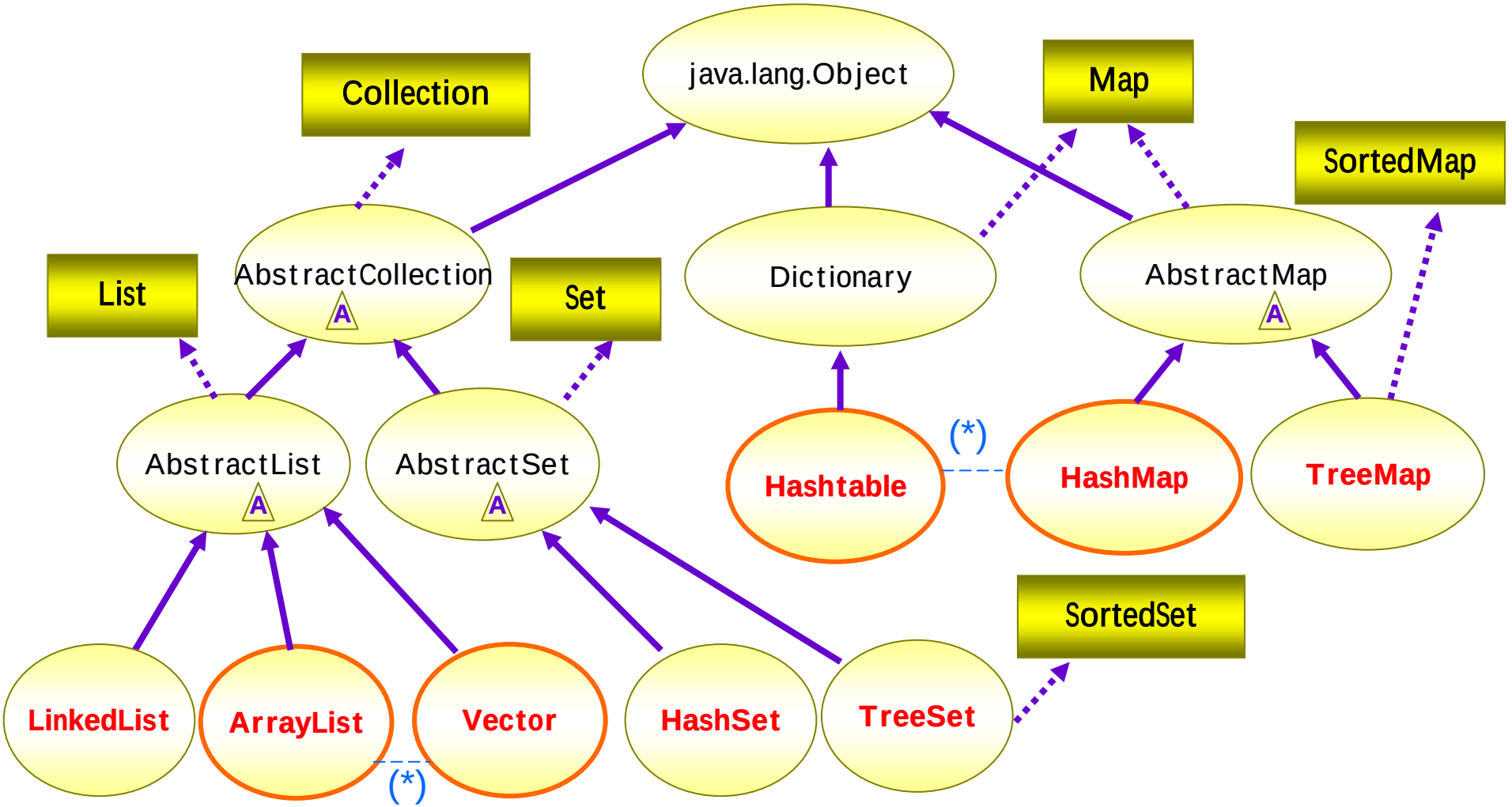
Java define interfaces como un medio para implementar estándares y guías. En el caso de colecciones usa 2 jerarquías de interfaces :



La interface **Collection** define un comportamiento genérico para colecciones sin restricciones, en donde los elementos pueden ser de cualquier tipo y pueden ser ordenadas/desordenadas, con duplicados o no, etc.

La interface **Map** define un comportamiento genérico para manejar **asociaciones arbitrarias de pares clave/valor**. No puede mantener claves duplicadas.00

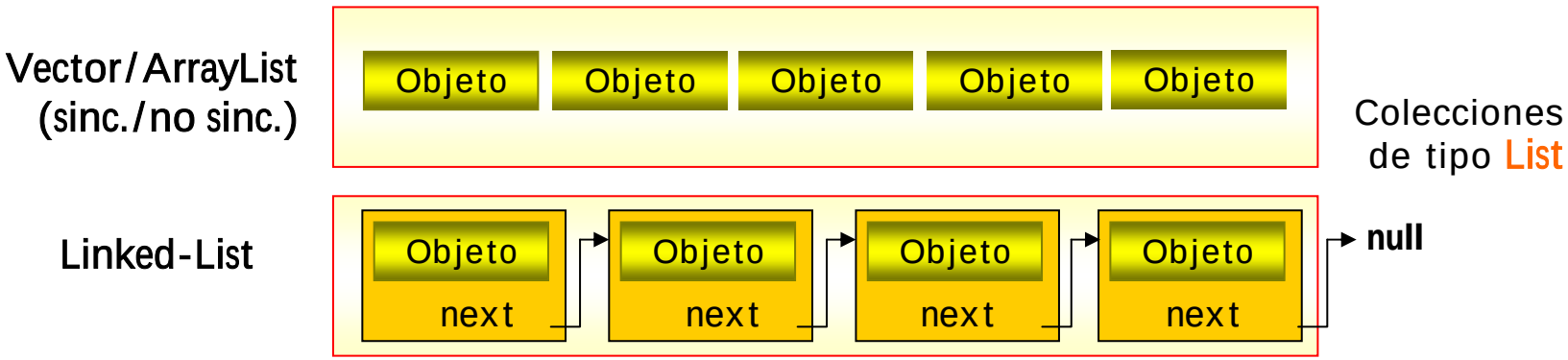
Clases del framework Collections



(*) HashMap/ArrayList son versiones de Hashtable/Vector respectivamente en donde sus métodos NO son sincronizados. Además Hashtable no permite nulos y HashMap si. ArrayList y Vector permiten nulos.

Tipos de Colecciones
La interface List

La interface **List** extiende la interface **Collection** para implementar una colección **indexada** de items. Las listas pueden ordenarse y pueden tener valores duplicados.

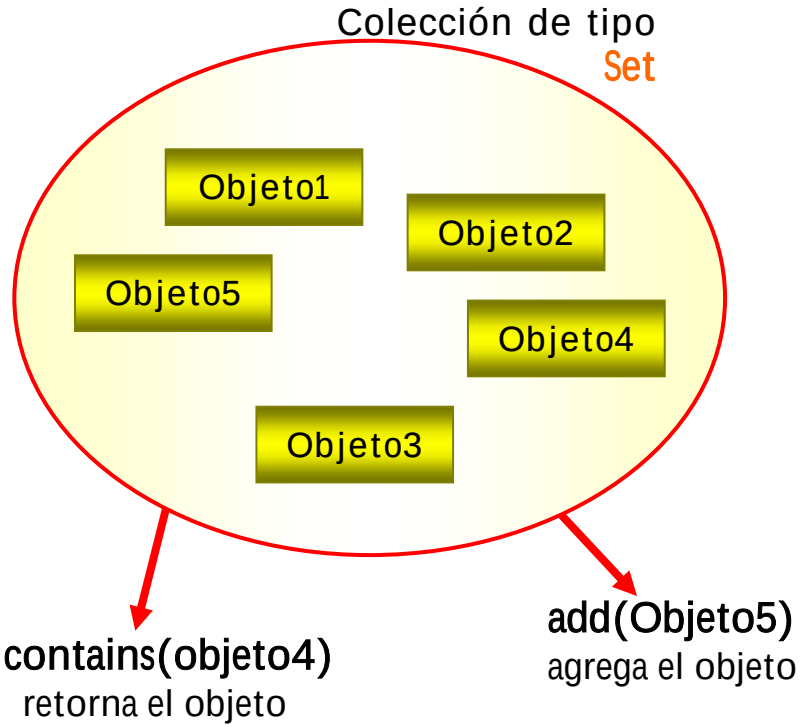



```
LinkedList actores = new LinkedList();
actores.add("Elizabeth Taylor");
actores.add("Richard Harris");
actores.add("Elizabeth Hurley");
actores.add("Richard Harris");
actores.add(0,"Zorrilla China");
System.out.println(actores);
-------------------------------------------------------------------------------
La salida es:
[Zorrilla China, Elizabeth Taylor, Richard Harris, Elizabeth Hurley, Richard Harris]
```

Valores duplicados

Tipos de Colecciones
La interface Set

La interface Set extiende la interface Collection para implementar una colección de items sin valores duplicados !!. La interface SortedSet extiende la interface Set para proveer un orden ascendente a sus elementos.




```
Set set = new HashSet();
set.add(new Integer(3));
set.add(new Integer(4));
set.add(new Long(60));
set.add(new Integer(3));
set.add(new Double(60.00));
set.add(new Float(70.00));
System.out.println(set);
----------------------------------------
La salida es: [60.0, 70.0, 60, 4, 3]
```

El orden no esta garantizado (implementa Set)

```
SortedSet instrumentos = new TreeSet();
instrumentos.add("Piano");
instrumentos.add("Saxo");
instrumentos.add("Violin");
instrumentos.add("Flauta");
System.out.println(instrumentos);
----------------------------------------
[Flauta, Piano, Saxo, Violin]
```

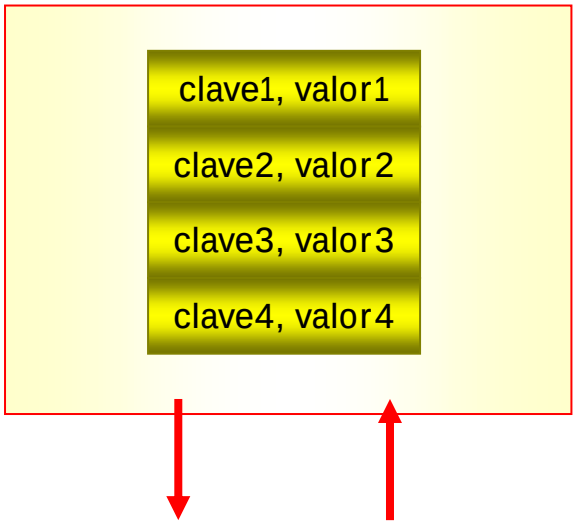
Orden Ascendente (implementa SortedSet)

Los objetos agregados a un SortedSet deben implementar la interface **Comparable**. En este caso los elementos agregados son de tipo **String** y esta clase lo provee.

Tipos de Colecciones
La interface Map

La interface Map provee métodos para manejar una colección formada por duplas: (clave,valor). Para buscar un objeto se debe contar con la clave. No exixten claves duplicadas !!!.

Colección de tipo Map



get(clave2) retorna el objeto.

put(clave1,valor1) agrega la dupla. Si la clave existe, reemplaza su valor.

- Para insertar objetos:

```
HashMap numer0s = new HashMap();
numeros.put(“uno", new Integer(1));
numeros.put(“dos”, new Integer(2));
numeros.put("tres", new Integer(3));
System.out.println(numeros)
-----------------------------------
La salida es: {dos=2, uno=1, tres=3}
```

El orden no esta garantizado (implementa Map)

Orden Ascendente (implementa SortedMap)

```
Con un TreeMap sería en orden:{dos=2, tres=3, uno=1}
```

- Para recuperar un objeto:

```
Integer n= (Integer)numeros.get(“dos");
if (n != null) {
    System.out.println(“El dos es="+n);
}
```

Se debe hacer un Casting Explícito al tipo deseado

Las claves de los objetos agregados a un **SortedMap** deben implementar la interface. Lo más común es que las claves sean de tipo **String,** la cual lo provee.

Tipos de Colecciones
Interfaces para recorrer (*Helpers*)

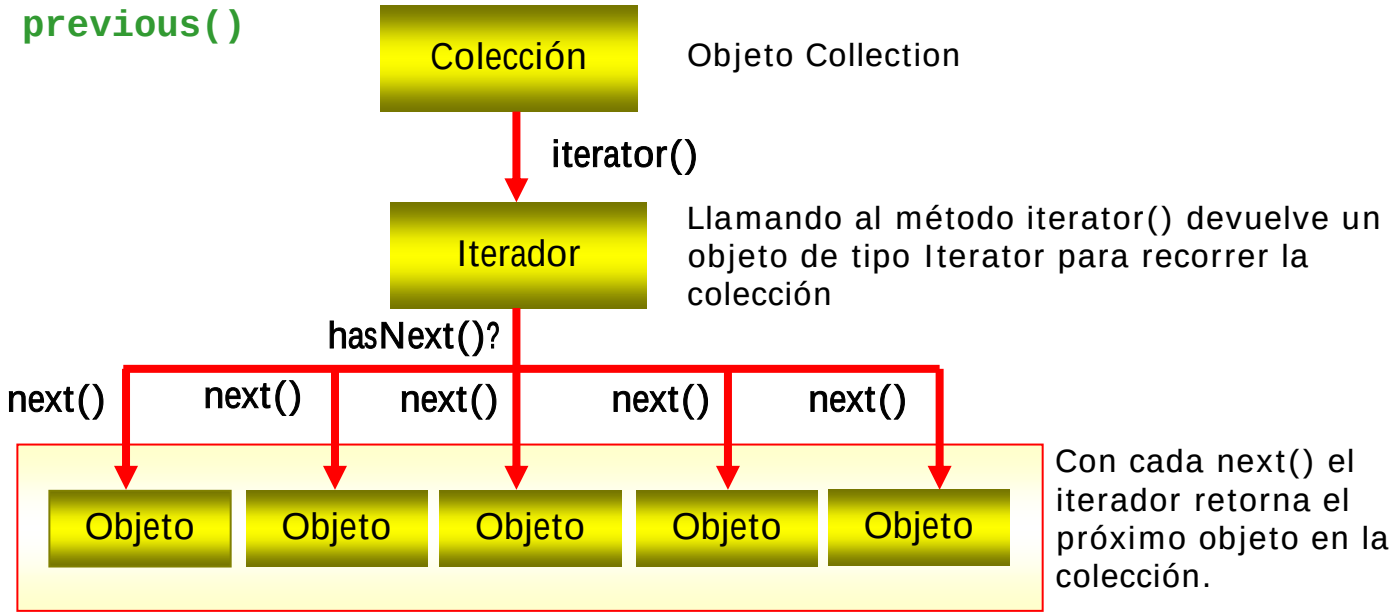
Además de estas interfaces, la API define un conjunto de interfaces para recorrer las colecciones y devolverlas en algún orden específico. Hay dos interfaces disponibles:

Iterator: Provee un mecanismo básico para recorrer la lista. Solamente se mueve hacia adelante. Los métodos de la interface son:

```
boolean hasnext()
Object next()
void remove()
```

ListIterator: Provee un mecanismo básico para recorrer la lista hacia adelante y hacia atrás. Además de los métodos de Iterator tiene :

```
boolean hasPrevious()
Object previous()
```



Tipos de Colecciones

La interfaces Iterator - *Un ejemplo*

Un iterador sirve para recorrer la colección una vez. Para recorrerlo otra vez, se debe obtener otro iterador.

```
Set instrumentos = new HashSet();
instrumentos.add("Piano");
instrumentos.add("Saxo");
instrumentos.add("Violin");
instrumentos.add("Flauta");
System.out.println("Se imprime con iterador:");
Iterator i = instrumentos.iterator();

while(i.hasNext()) {
    System.out.println(i.next());
}

System.out.println("Se imprime la colección:");
System.out.println(instrumentos);
```

devuelve true: si hay mas elementos
false: si se llegó al final de la colección

devuelve el próximo objeto

Salida:

```
Se imprime con iterador:
Violín
Piano
Flauta
Saxo
Se imprime la colección:
[Violin, Piano, Flauta, Saxo]
```

La interface Iterador, además de estos 2 métodos tiene el método **`remove()`** que permite eliminar de la colección el último elemento retornado por la iteración.

Tipos de Colecciones

La interfaces ListIterator - *Un ejemplo*

Un ListIterador permite recorrer la colección más una vez, en un sentido y luego en otro y permite modificar la colección duante el recorrido.

```
LinkedList animales = new LinkedList();
animales.add("Perro");
animales.add("Gato");
animales.add("Conejo");
animales.add("Tortuga");
animales.add("Lechuza");
System.out.println("Se imprime con iterador");

ListIterator iter=animales.listIterator(animales.size());

while(iter.hasPrevious()){
      System.out.println(iter.previous());
}

System.out.println("Se imprime la colección:");
System.out.println(animales);
```

Retorna un iterador que puede recorrer la colección en ambas direcciones

devuelve **true**: si hay elementos anteriores

false: si se llegó al inicio de la colección

devuelve el objeto anterior

Salida:

```
Se imprime con iterador:
Lechuza
Tortuga
Conejo
Gato
Perro

Se imprime la colección:
[Perro, Gato, Conejo,
Tortuga, Lechuza]
```

Colecciones
Funcionalidades de comparación

1) Implementando la interface java.lang.Comparable. Esta *interface* es implementada para darle a una clase un orden natural. Dada una colección de objeto del mismo tipo, la interface permite ordenar la colección en un orden natural.

```
public interface Comparable {
  public int compareTo(Object o);
}
```

Este método toma otro Object como argumento y retorna:
- **valor negativo** si el objeto actual es > el argumento (o se ubica después)
- **0** si el argumento es igual
- **valor positivo** si el objeto actual es menor que el argumento (o se ubica antes)

Dentro de la API Java 2 SDK, versión 1.2, hay más de 15 clases que implementan la *interface* Comparable.

```
public class Alumno implements Comparable {
 private String apeynom;
 int legajo;


  public int compareTo(Object objetoAComparar){
    Alumno a = (Alumno) objetoAComparar;
     return (this.legajo - a.legajo);
  }
}
```

```
TreeSet alumnos = new TreeSet();
alumnos.add(new Alumno("Juan", 24607));
. . .
```

Cuando se hace el add(), chequea que el objeto a insertar haya implementado la interface Comparable.. De lo contrario, dispara una Excepción en *run-time. De tipo* **java.lang.ClassCastException**

Colecciones
Funcionalidades de comparación

2) El otro mecanimso le provee a la colección un objeto que implementa la interface java.lang.Comparator.

```
public interface Comparator{
  public int compare(Object obj1, Object obj2);
}
```

Este método toma otro Object como argumento y retorna:
- **valor negativo** si el objeto obj1 es < obj2 (o se ubica antes)
- **0** si obj1 es igual obj2
- **valor positivo** si el objeto obj1 es > obj2 (o se ubica después)

Dispara una excpeción de tipo **`java.lang.ClassCastException`** en run-time si los objetos son incomparables.

Ejemplo:

```
public class AlumnoComparator implements Comparator{
  public int compare(Object o1, Object o2) {
    return ((Alumno)o1).legajo-((Alumno)o2).legajo;
  }
}
```

```
TreeSet alumnos  =  new TreeSet(new AlumnoComparator());
alumnos.add(new Alumno("Juan", 24607));
alumnos.add(new Alumno("Maria", 23707));
alumnos.add(new String("Pedro"));
```

→ **`java.lang.ClassCastException`**

Colecciones

SortedMap con claves de un tipo distinto a String

```
import java.util.Iterator;
import java.util.TreeMap;

public class Colecciones {

public static void main(String[] args){

  SortedMap tabla = new TreeMap();
  tabla.put(new Alumno("Juan", 30), "30");
  tabla.put(new Alumno("Maria", 110), "110");
  tabla.put(new Alumno("Pedro", 70), "70");
  tabla.put(new Alumno("Sol", 1), "110");
  tabla.put(new Alumno("Ana", 999), "70");

  Iterator iter = (Iterator) tabla.keySet().iterator();
    while (iter.hasNext()) {
      Alumno element = (Alumno) iter.next();
      System.out.println("Alumno con cast:"+element);
  }

  iter = (Iterator) tabla.keySet().iterator();
  while (iter.hasNext()) {
      System.out.println(iter.next());
  }
 }
}
```

La clave es un objeto de tipo Alumno

Se invoca al método toString() automáticamente.

```
public class Alumno implements Comparable {
private String apeynom;
private int legajo;

public Alumno(String a, int l){
   apeynom = a;
   legajo=l;
}

public int compareTo(Object objetoAComparar){
   Alumno a = (Alumno) objetoAComparar;
   return (this.legajo - a.legajo);
 }

 public String toString(){
   return
    "Alumno"+apeynom+"delegajo:"+legajo+"\n";
 }
}
```

```
Alumno con cast:Alumno Sol de legajo: 1
Alumno con cast:Alumno Juan de legajo: 30
Alumno con cast:Alumno Pedro de legajo: 70
Alumno con cast:Alumno Maria de legajo: 110
Alumno con cast:Alumno Ana de legajo: 999

Alumno Sol de legajo: 1
Alumno Juan de legajo: 30
Alumno Pedro de legajo: 70
Alumno Maria de legajo: 110
Alumno Ana de legajo: 999
```

SALIDA

El framework Collections

Para más información, consulte el artículo titulado "Introduction to the Collections Framework" en el siguiente URL:
http://developer.Java.sun.com/developer/onlineTraining/collections/